J. Parallel Distrib. Comput. 73 (2013) 42–51

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Enhancing data parallelism for Ant Colony Optimization on GPUs

José M. Cecilia[a,*], José M. García[a], Andy Nisbet[b], Martyn Amos[b], Manuel Ujaldón[c]

[a] Computer Architecture Department, University of Murcia, 30100 Murcia, Spain
[b] Novel Computation Group, Division of Computing and IS, Manchester Metropolitan University, Manchester M1 5GD, UK
[c] Computer Architecture Department, University of Malaga, 29071 Málaga, Spain

ARTICLE INFO

Article history:
Received 12 April 2011
Received in revised form
2 January 2012
Accepted 9 January 2012
Available online 20 January 2012

Keywords:
Metaheuristics
GPU programming
Ant Colony Optimization
TSP
Performance analysis

ABSTRACT

Graphics Processing Units (GPUs) have evolved into highly parallel and fully programmable architecture over the past five years, and the advent of CUDA has facilitated their application to many real-world applications. In this paper, we deal with a GPU implementation of Ant Colony Optimization (ACO), a population-based optimization method which comprises two major stages: *tour construction* and *pheromone update*. Because of its inherently parallel nature, ACO is well-suited to GPU implementation, but it also poses significant challenges due to irregular memory access patterns. Our contribution within this context is threefold: (1) a data parallelism scheme for *tour construction* tailored to GPUs, (2) novel GPU programming strategies for the *pheromone update* stage, and (3) a new mechanism called I-Roulette to replicate the classic roulette wheel while improving GPU parallelism. Our implementation leads to factor gains exceeding 20x for any of the two stages of the ACO algorithm as applied to the TSP when compared to its sequential counterpart version running on a similar single-threaded high-end CPU. Moreover, an extensive discussion focused on different implementation paths on GPUs shows the way to deal with parallel graph connected components. This, in turn, suggests a broader area of inquiry, where algorithm designers may learn to adapt similar optimization methods to GPU architecture.

1. Introduction

Ant Colony Optimization (ACO) [12] is a population-based search method inspired by the behavior of real ants. It may be applied to a wide range of problems [8,2], many of which are graph-theoretic in nature. It was first applied to the Traveling Salesman Problem (TSP) [19] by Dorigo et al. [10,11].

In essence, simulated ants construct solutions to the TSP in the form of *tours*. The artificial ants are simple agents which construct tours in a parallel, probabilistic fashion. They are guided in this task by simulated *pheromone trails* and *heuristic information*. Pheromone trails are a fundamental component of the algorithm, since they facilitate indirect communication between agents via their *environment*, a process known as *stigmergy* [9]. For additional details about these processes, we refer the reader to [12].

ACO algorithms are population-based, that is, a collection of agents "collaborates" to find an optimal (or at least satisfactory) solution. Such approaches are suited to parallel processing, but their success strongly depends on the nature of the particular problem and the underlying hardware available. Several parallelization strategies have been proposed for the ACO algorithm on shared and distributed memory architecture [28,18,21].

The *Graphics Processing Unit* (GPU) is a topic of significant interest in high performance computing. For applications with abundant parallelism, GPUs deliver higher peak computational throughput than latency-oriented CPUs, thus offering a tremendous potential performance uplift on massively parallel problems [14]. Of particular relevance to us are attempts to parallelize the ACO algorithm on GPUs. Until now, these approaches have focused on accelerating the *tour construction*, step performed by each ant by taking a task-based parallelism approach, with pheromone deposition on the CPU [13,3,20].

In this paper, we present the first fully developed ACO algorithm for the Traveling Salesman Problem (TSP) on GPUs, where *both* stages are parallelized: *tour construction* and *pheromone update*. A *data parallelism* approach, which is better suited to the GPU parallelism model, is used to enhance performance on the first stage, and several GPU design patterns are evaluated for the parallelization of the second stage. Our major contributions include the following:

1. To the best of our knowledge, this is the first data-parallelism scheme on GPUs for the ACO tour construction stage. Our design proposes two different types of virtual ants: *Queen*

* Corresponding author.
E-mail addresses: chema@ditec.um.e (J.M. Cecilia), jmgarcia@ditec.um.ess (J.M. García), a.nisbet@mmu.ac.uk (A. Nisbet), m.amos@mmu.ac.u (M. Amos), ujaldon@uma.es (M. Ujaldón).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.01.002

ants (associated with CUDA thread-blocks), and *worker* ants (associated with CUDA threads).
2. We introduce an I-*Roulette* method (Independent Roulette) to replicate the classic roulette wheel selection while improving GPU parallelism.
3. We discuss the implementation of the pheromone update stage on GPUs, using either atomic operations or other GPU alternatives.
4. We offer an in-depth analysis of both stages of the ACO algorithm for different instances of the TSP problem. Several GPU parameters are tuned to reach a speed-up factor of up to 21× for the tour construction stage, and a 20× speed-up factor for the pheromone update stage.
5. The solution accuracy obtained by our GPU algorithms is compared to that of the sequential code given in [12] and extended using TSPLIB.

The rest of the paper is organized as follows. We briefly introduce Ant Colony Optimization for the TSP and Compute Unified Device Architecture (CUDA) from NVIDIA in Section 2. In Section 3 we present GPU designs for the main stages of the ACO algorithm. Our experimental methodology is outlined in Section 4 before we describe the performance evaluation of our algorithm in Section 5. Other parallelization strategies for the ACO algorithm are described in Section 6, before we summarize our findings and conclude with suggestions for future work.

2. Background

2.1. Ant Colony Optimization for the traveling salesman problem

The Traveling Salesman Problem (TSP) [19] involves finding the shortest (or "cheapest") round-trip route that visits each of a number of "cities" exactly once. The symmetric TSP on n cities may be represented as a complete weighted graph, G, of n nodes, with each weighted edge, $e_{i,j}$, representing the inter-city distance $d_{i,j} = d_{j,i}$ between cities i and j. The TSP is a well-known NP-hard optimization problem, and is used as a standard benchmark for many heuristic algorithms [17].

The TSP was the first problem solved by Ant Colony Optimization (ACO) [11,7]. This method uses a number of simulated "ants" (or *agents*), which perform distributed search on a graph. Each ant moves on the graph until it completes a tour, and then offers this tour as its suggested solution. In order to achieve this latter step, each ant drops "pheromone" on the edges that it visits during its tour. The quantity of pheromone dropped, if any, is determined by the *quality* of the ant's solution relative to those obtained by the other ants. The ants probabilistically choose the next city to visit, based on *heuristic information* obtained from inter-city distances and the net pheromone trail. Although such heuristic information drives the ants toward an optimal solution, a process of pheromone "evaporation" is also applied in order to prevent the process stalling in a local minimum.

The Ant System (AS) is an early variant of ACO, first proposed by Dorigo [7]. The AS algorithm is divided into two main stages: *tour construction* and *pheromone update*. Tour construction is based on m ants building tours in parallel. Initially, ants are randomly placed. At each construction step, each ant applies a probabilistic action choice rule, called the *random proportional rule*, which decides the city to visit next. The probability for ant k, placed at city i, of visiting city j is given by the Eq. (1)

$$p^k_{i,j} = \frac{\left[\tau_{i,j}\right]^{\alpha}\left[\eta_{i,j}\right]^{\beta}}{\sum\limits_{l \in N^k_i}\left[\tau_{i,l}\right]^{\alpha}\left[\eta_{i,l}\right]^{\beta}}, \quad \text{if } j \in N^k_i, \tag{1}$$

where $\eta_{i,j} = 1/d_{i,j}$ is a heuristic value determined *a priori*, α and β are two parameters determining the relative *influences* of the pheromone trail and the heuristic information respectively, and N^k_i is the feasible neighborhood of ant k when at city i. This latter set represents the set of cities that ant k has not yet visited; the probability of choosing a city outside N^k_i is zero (this prevents an ant returning to a city, which is not allowed in the TSP). By this probabilistic rule, the probability of choosing a particular edge (i, j) increases with the value of the associated pheromone trail $\tau_{i,j}$ and of the heuristic information value $\eta_{i,j}$. The numerator of the Eq. (1) is the same for every ant in a single run, which encourages efficiency by storing this information in an additional matrix, called *choice_info* (see [12]). The random proportional rule ends with a selection procedure, which is done analogously to the *roulette wheel* selection procedure of evolutionary computation (see [12,15]). Each value *choice_info*[*current_city*][*j*] of a city j that ant k has not yet visited is associated with a slice on a circular roulette wheel, with the size of the slice being proportional to the weight of the associated choice. The wheel is then "spun", and the city to which a fixed marker points is chosen as the next city for ant k. Additionally, each ant k maintains a memory, M^k, called the *tabu list*, which contains a chronological ordering of the cities already visited. This memory is used to determine the feasible neighborhood, and also allows an ant to (1) compute the length of the tour T^k it generated, and (2) retrace the path to deposit pheromone.

After all ants have constructed their tours, the pheromone trails are updated. This is achieved by first lowering the pheromone value on all edges by a constant factor (analogous to evaporation), and then adding pheromone to edges that ants have crossed in their tours. Pheromone evaporation is implemented by

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}, \quad \forall (i,j) \in L, \tag{2}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate. After evaporation, all ants deposit pheromone on their visited edges:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau^k_{i,j}, \quad \forall (i,j) \in L, \tag{3}$$

where $\Delta\tau_{ij}$ is the amount of pheromone ant k deposits. This is defined as follows:

$$\Delta\tau^k_{i,j} = \begin{cases} 1/C^k & \text{if } e(i,j)^k \text{ belongs to } T^k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where C^k, the length of the tour T^k built by the k-th ant, is computed as the sum of the lengths of the edges belonging to T^k. According to Eq. (4), the better an ant's tour, the more pheromone the edges belonging to this tour receive. In general, edges that are used by many ants (and which are part of short tours), receive more pheromone, and are therefore more likely to be chosen by ants in future iterations of the algorithm.

2.2. The CUDA programming model

All Nvidia GPU platforms from the G80 architecture may be programmed using the Compute Unified Device Architecture (CUDA) programming model, which makes GPUs operate as a highly parallel computing device. Each GPU device is a scalable processor array consisting of a set of SIMT (Single Instruction Multiple Threads) Streaming Multiprocessors (SM), each containing several stream processors (SPs). Different memory spaces are available in each GPU on the system. The global memory (also called *device* or video memory) is the only space accessible to all multiprocessors. It is the largest (and slowest) memory space, and is private to each GPU on the system. Additionally,